


```
PPPPPPPP      AAAAAA      SSSSSSSS      RRRRRRRR      EEEEEEEEEE      AAAAAA      DDDDDDDD      UU      UU      TTTTTTTTTT
PPPPPPPP      AAAAAA      SSSSSSSS      RRRRRRRR      EEEEEEEEEE      AAAAAA      DDDDDDDD      UU      UU      TTTTTTTTTT
PP      PP      AA      AA      SS      RR      RR      EE      AA      AA      DD      DD      UU      UU      TT
PP      PP      AA      AA      SS      RR      RR      EE      AA      AA      DD      DD      UU      UU      TT
PP      PP      AA      AA      SS      RR      RR      EE      AA      AA      DD      DD      UU      UU      TT
PPPPPPPP      AA      AA      SSSSSS      RRRRRRRR      EEEEEEEE      AA      AA      DD      DD      UU      UU      TT
PPPPPPPP      AA      AA      SSSSSS      RRRRRRRR      EEEEEEEE      AA      AA      DD      DD      UU      UU      TT
PP      AAAAAAAAAA      SS      RR      RR      EE      AAAAAAAAAA      DD      DD      UU      UU      TT
PP      AAAAAAAAAA      SS      RR      RR      EE      AAAAAAAAAA      DD      DD      UU      UU      TT
PP      AA      AA      SS      RR      RR      EE      AA      AA      DD      DD      UU      UU      TT
PP      AA      AA      SSSSSSSS      RR      RR      EEEEEEEEEE      AA      AA      DDDDDDDD      UUUUUUUUUU      TT
PP      AA      AA      SSSSSSSS      RR      RR      EEEEEEEEEE      AA      AA      DDDDDDDD      UUUUUUUUUU      TT
                                          ....
                                          ....
                                          ....
                                          ....
```

```
LL      IIIIII      SSSSSSSS
LL      IIIIII      SSSSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SSSSSS
LL      II      SSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SS
LLLLLLLLLL      IIIIII      SSSSSSSS
LLLLLLLLLL      IIIIII      SSSSSSSS
```



```
1 0001 0 MODULE PAS$$READ_UTIL ( %TITLE, 'Utility routines used by READ'
2 0002 0 IDENT = '1-001' ! File: PASREADUT.B32 Edit: SBL1001
3 0003 0 ) =
4 0004 1 BEGIN
5 0005 1
6 0006 1 *****
7 0007 1 *
8 0008 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
9 0009 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
10 0010 1 * ALL RIGHTS RESERVED.
11 0011 1 *
12 0012 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
13 0013 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
14 0014 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
15 0015 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
16 0016 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
17 0017 1 * TRANSFERRED.
18 0018 1 *
19 0019 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
20 0020 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
21 0021 1 * CORPORATION.
22 0022 1 *
23 0023 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
24 0024 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
25 0025 1 *
26 0026 1 *
27 0027 1 *****
28 0028 1
29 0029 1
30 0030 1 ++
31 0031 1 FACILITY: Pascal Language Support
32 0032 1
33 0033 1 ABSTRACT:
34 0034 1
35 0035 1 This module contains utility procedures used by
36 0036 1 the numeric READ procedures.
37 0037 1
38 0038 1 ENVIRONMENT: User mode - AST reentrant
39 0039 1
40 0040 1 AUTHOR: Steven B. Lionel, CREATION DATE: 1-April-1981
41 0041 1
42 0042 1 MODIFIED BY:
43 0043 1
44 0044 1 1-001 - Original. SBL 1-April-1981
45 0045 1 --
46 0046 1
```

```
48 0047 1 %SBTTL 'Declarations'
49 0048 1
50 0049 1 PROLOGUE DEFINITIONS:
51 0050 1
52 0051 1
53 0052 1 REQUIRE 'RTLIN:PASPROLOG';           ! Externals, linkages, PSECTs, structures
54 0116 1
55 0117 1
56 0118 1 TABLE OF CONTENTS:
57 0119 1
58 0120 1
59 0121 1 FORWARD ROUTINE
60 0122 1 PASS$GET_UNSIGNED: JSB READ UTIL,      ! Get an unsigned string
61 0123 1 PASS$GET_INTEGER: JSB READ UTIL,      ! Get an integer string
62 0124 1 PASS$GET_REAL: JSB READ UTIL,         ! Get a real string
63 0125 1 PASS$GET_ENUMERATED: JSB_READ_UTIL,   ! Get an enumerated value string
64 0126 1
65 0127 1 FIND_NON_BLANK: JSB_FIND_NON_BLANK;   ! Find next non-blank character
66 0128 1
67 0129 1
68 0130 1 MACROS:
69 0131 1
70 0132 1 NONE
71 0133 1
72 0134 1 EQUATED SYMBOLS:
73 0135 1
74 0136 1
75 0137 1 LITERAL
76 0138 1 !+
77 0139 1 ! Character class codes used below for CLASSTAB.
78 0140 1 !-
79 0141 1
80 0142 1 iv = 0,      ! Invalid character
81 0143 1 BT = 1,      ! Blank or Tab
82 0144 1 DG = 2,      ! Digit
83 0145 1 DP = 3,      ! Decimal Point
84 0146 1 SI = 4,      ! Sign
85 0147 1 EL = 5,      ! Exponent letter
86 0148 1 LT = 6,      ! Other letter, dollar and underscore
87 0149 1
88 0150 1 !+
89 0151 1 ! Aliases for class codes for use in routines.
90 0152 1 !-
91 0153 1
92 0154 1 CLASS_IV = iv,
93 0155 1 CLASS_BT = BT,
94 0156 1 CLASS_DG = DG,
95 0157 1 CLASS_DP = DP,
96 0158 1 CLASS_SI = SI,
97 0159 1 CLASS_EL = EL,
98 0160 1 CLASS_LT = LT;
99 0161 1
100 0162 1
101 0163 1 FIELDS:
102 0164 1
103 0165 1 NONE
104 0166 1
```



```

: 105      0167 1 ! OWN STORAGE:
: 106      0168 1 !
: 107      0169 1 !
: 108      0170 1 OWN
: 109      0171 1
: 110      0172 1
: 111      0173 1
: 112      0174 1
: 113      0175 1
: 114      0176 1
: 115      0177 1
: 116      0178 1
: 117      0179 1
: 118      0180 1
: 119      0181 1
: 120      0182 1
: 121      0183 1
: 122      0184 1
: 123      0185 1
: 124      0186 1
: 125      0187 1
: 126      0188 1
: 127      0189 1

!+ The following table is used for determining the class of a particular
! character. Each of the first 128 characters is assigned a class code
! as listed above in the LITERAL section.
!-

CLASSTAB: VECTOR [128, BYTE] PSECT (_PASSCODE) INITIAL (BYTE(
    iv,iv,iv,iv,iv,iv,iv,iv,iv,iv,BT,iv,iv,iv,iv,iv,iv,    ! 00-0F
    iv,iv,iv,iv,iv,iv,iv,iv,iv,iv,iv,iv,iv,iv,iv,iv,    ! 10-1F
    BT,iv,iv,iv,LT,iv,iv,iv,iv,iv,iv,SI,iv,SI,DP,iv,    ! 20-2F
    DG,DG,DG,DG,DG,DG,DG,DG,DG,DG,iv,iv,iv,iv,iv,iv,    ! 30-3F
    iv,LT,LT,LT,EL,EL,LT,LT,LT,LT,LT,LT,LT,LT,LT,LT,    ! 40-4F
    LT,EL,LT,LT,LT,LT,LT,LT,LT,LT,LT,iv,iv,iv,iv,LT,    ! 50-5F
    iv,LT,LT,LT,EL,EL,LT,LT,LT,LT,LT,LT,LT,LT,LT,LT,    ! 60-6F
    LT,EL,LT,LT,LT,LT,LT,LT,LT,LT,LT,LT,iv,iv,iv,iv));    ! 70-7F

```

```
: 129      0190 1 %SBTTL 'PAS$$GET_UNSIGNED - Find an unsigned number string'
: 130      0191 1 GLOBAL ROUTINE PAS$$GET_UNSIGNED (      | Get unsigned number string
: 131      0192 1      PFV: REF $PAS$PFV FILE_VARIABLE,    | Pascal File Variable
: 132      0193 1      IN_FCB: REF $PAS$FCB_CONTROL_BLOCK; | File control block
: 133      0194 1      STRING_ADDR,                      | Output string address
: 134      0195 1      STRING_LEN,                       | Output string length
: 135      0196 1      FCB: REF $PAS$FCB_CONTROL_BLOCK     | File control block
: 136      0197 1      ) : JSB_READ_UTIL =
: 137      0198 1
: 138      0199 1 ++
: 139      0200 1 FUNCTIONAL DESCRIPTION:
: 140      0201 1
: 141      0202 1     This procedure advances the textfile referenced by FCB until it
: 142      0203 1     locates a string that satisfies the Pascal UNSIGNED datatype
: 143      0204 1     syntax. The address and length of that string are returned as
: 144      0205 1     output parameters.
: 145      0206 1
: 146      0207 1 CALLING SEQUENCE:
: 147      0208 1
: 148      0209 1     Valid.wc.v = JSB_READ_UTIL PAS$$GET_UNSIGNED (PFV.mr.r, IN_FCB.mr.r;
: 149      0210 1     STRING_ADDR.wl.v, STRING_LEN.wl.v, FCB.mr.r)
: 150      0211 1
: 151      0212 1 FORMAL PARAMETERS:
: 152      0213 1
: 153      0214 1     PFV          - The Pascal File Variable of the file.
: 154      0215 1
: 155      0216 1     IN_FCB       - The File Control Block of the file being scanned.
: 156      0217 1     It is assumed to be a textfile.
: 157      0218 1
: 158      0219 1     STRING_ADDR  - Output register parameter which is set to the
: 159      0220 1     address of the first byte of the string.
: 160      0221 1
: 161      0222 1     STRING_LEN   - Output register parameter which is set to the
: 162      0223 1     length of the string in bytes.
: 163      0224 1
: 164      0225 1     FCB          - Output register parameter which is the same as IN_FCB.
: 165      0226 1
: 166      0227 1 IMPLICIT INPUTS:
: 167      0228 1
: 168      0229 1     It is assumed that lazy-lookahead is not in progress.
: 169      0230 1
: 170      0231 1 IMPLICIT OUTPUTS:
: 171      0232 1
: 172      0233 1     FCB$A_RECORD_CUR points to the next character after the string, or
: 173      0234 1     EOL.
: 174      0235 1
: 175      0236 1 ROUTINE VALUE:
: 176      0237 1
: 177      0238 1     1 if string is a valid unsigned, 0 otherwise
: 178      0239 1     If 0 is returned, the pointer and length include the first bad character.
: 179      0240 1
: 180      0241 1 SIDE EFFECTS:
: 181      0242 1
: 182      0243 1     NONE
: 183      0244 1
: 184      0245 1 SIGNALLED ERRORS:
: 185      0246 1
```



```
186 0247 1 | NONE
187 0248 1 |
188 0249 1 | --
189 0250 1 |
190 0251 2 BEGIN
191 0252 2
192 0253 2 LOCAL
193 0254 2 CHAR; ! Character read
194 0255 2
195 0256 2 | +
196 0257 2 | Declare CHAR_BYTE which is the same as CHAR except that we can
197 0258 2 | test it as a signed byte. We want to leave CHAR as a longword
198 0259 2 | so that it can be used efficiently as an index.
199 0260 2 | -
200 0261 2
201 0262 2 BIND
202 0263 2 CHAR_BYTE = CHAR: BYTE SIGNED;
203 0264 2
204 0265 2 | +
205 0266 2 | Find first character that is not a blank or a tab, possibly skipping
206 0267 2 | records.
207 0268 2 | -
208 0269 2
209 0270 2 CHAR = FIND_NON_BLANK (PFV [PFV$R_PFV], IN_FCB [FCB$R_FCB]; FCB);
210 0271 2
211 0272 2 | +
212 0273 2 | At this point, CHAR contains the first character which is not a blank
213 0274 2 | or a tab. Initialize STRING_ADDR.
214 0275 2 | -
215 0276 2
216 0277 2 STRING_ADDR = .FCB [FCB$A_RECORD_CUR];
217 0278 2
218 0279 2 | +
219 0280 2 | In a loop, classify the characters until end-of-line or an invalid
220 0281 2 | character is found.
221 0282 2 | -
222 0283 2
223 0284 2 WHILE 1 DO
224 0285 2 BEGIN
225 0286 2
226 0287 2 | +
227 0288 2 | Screen out characters 128-255, which are not in CLASSTAB, by
228 0289 2 | doing a signed byte test for a negative value.
229 0290 2 | -
230 0291 2
231 0292 2 IF .CHAR_BYTE LSS 0
232 0293 2 THEN
233 0294 2 EXITLOOP;
234 0295 2
235 0296 2 | +
236 0297 2 | If the character is not a digit, exit.
237 0298 2 | -
238 0299 2
239 0300 2 IF .CLASSTAB [.CHAR] NEQU CLASS_DG
240 0301 2 THEN
241 0302 2 EXITLOOP;
242 0303 2
```

PC	OP	OP2	OP3	OP4	OP5	OP6	OP7	OP8	OP9	OP10	OP11	OP12	OP13	OP14	OP15	OP16	OP17	OP18	OP19	OP20	OP21	OP22	OP23	OP24	OP25	OP26	OP27	OP28	OP29	OP30	OP31	OP32	OP33	OP34	OP35	OP36	OP37	OP38	OP39	OP40	OP41	OP42	OP43	OP44	OP45	OP46	OP47	OP48	OP49	OP50	OP51	OP52	OP53	OP54	OP55	OP56	OP57	OP58	OP59	OP60	OP61	OP62	OP63	OP64	OP65	OP66	OP67	OP68	OP69	OP70	OP71	OP72	OP73	OP74	OP75	OP76	OP77	OP78	OP79	OP80	OP81	OP82	OP83	OP84	OP85	OP86	OP87	OP88	OP89	OP90	OP91	OP92	OP93	OP94	OP95	OP96	OP97	OP98	OP99	OP100	OP101	OP102	OP103	OP104	OP105	OP106	OP107	OP108	OP109	OP110	OP111	OP112	OP113	OP114	OP115	OP116	OP117	OP118	OP119	OP120	OP121	OP122	OP123	OP124	OP125	OP126	OP127	OP128	OP129	OP130	OP131	OP132	OP133	OP134	OP135	OP136	OP137	OP138	OP139	OP140	OP141	OP142	OP143	OP144	OP145	OP146	OP147	OP148	OP149	OP150	OP151	OP152	OP153	OP154	OP155	OP156	OP157	OP158	OP159	OP160	OP161	OP162	OP163	OP164	OP165	OP166	OP167	OP168	OP169	OP170	OP171	OP172	OP173	OP174	OP175	OP176	OP177	OP178	OP179	OP180	OP181	OP182	OP183	OP184	OP185	OP186	OP187	OP188	OP189	OP190	OP191	OP192	OP193	OP194	OP195	OP196	OP197	OP198	OP199	OP200	OP201	OP202	OP203	OP204	OP205	OP206	OP207	OP208	OP209	OP210	OP211	OP212	OP213	OP214	OP215	OP216	OP217	OP218	OP219	OP220	OP221	OP222	OP223	OP224	OP225	OP226	OP227	OP228	OP229	OP230	OP231	OP232	OP233	OP234	OP235	OP236	OP237	OP238	OP239	OP240	OP241	OP242	OP243	OP244	OP245	OP246	OP247	OP248	OP249	OP250	OP251	OP252	OP253	OP254	OP255	OP256	OP257	OP258	OP259	OP260	OP261	OP262	OP263	OP264	OP265	OP266	OP267	OP268	OP269	OP270	OP271	OP272	OP273	OP274	OP275	OP276	OP277	OP278	OP279	OP280	OP281	OP282	OP283	OP284	OP285	OP286	OP287	OP288	OP289	OP290	OP291	OP292	OP293	OP294	OP295	OP296	OP297	OP298	OP299	OP300	OP301	OP302	OP303	OP304	OP305	OP306	OP307	OP308	OP309	OP310	OP311	OP312	OP313	OP314	OP315	OP316	OP317	OP318	OP319	OP320	OP321	OP322	OP323	OP324	OP325	OP326	OP327	OP328	OP329	OP330	OP331	OP332	OP333	OP334	OP335	OP336	OP337	OP338	OP339	OP340	OP341	OP342	OP343	OP344	OP345	OP346	OP347	OP348	OP349	OP350	OP351	OP352	OP353	OP354	OP355	OP356	OP357	OP358	OP359	OP360	OP361	OP362	OP363	OP364	OP365	OP366	OP367	OP368	OP369	OP370	OP371	OP372	OP373	OP374	OP375	OP376	OP377	OP378	OP379	OP380	OP381	OP382	OP383	OP384	OP385	OP386	OP387	OP388	OP389	OP390	OP391	OP392	OP393	OP394	OP395	OP396	OP397	OP398	OP399	OP400	OP401	OP402	OP403	OP404	OP405	OP406	OP407	OP408	OP409	OP410	OP411	OP412	OP413	OP414	OP415	OP416	OP417	OP418	OP419
----	----	-----	-----	-----	-----	-----	-----	-----	-----	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

PAS\$\$READ_UTIL		Utility routines used by READ		F 5		16-Sep-1984 01:55:25		VAX-11 Bliss-32 V4.0-742		Page 7	
1-001		PAS\$\$GET_UNSIGNED - Find an unsigned number str		14-Sep-1984 12:51:47				[PASRTL.SRC]PASREADUT.B32;1		(3)	

			EC	A7	D6	00013		INCL	-20(FCB)		: 0308
	F0	A7	EC	A7	D1	00016		CMPL	-20(FCB), -16(FCB)		: 0309
				06	1E	0001B		BGEQU	2\$:
		50	EC	B7	9A	0001D		MOVZBL	@-20(FCB), CHAR		: 0311
				E4	11	00021		BRB	1\$:
55		EC	A7	54	C3	00023	2\$:	SUBL3	STRING_ADDR, -20(FCB), STRING_LEN		: 0322
				04	13	00028		BEQL	3\$: 0323
		50		01	D0	0002A		MOVL	#1, R0		: 0325
		55			05	0002D		RSB			:
				01	D0	0002E	3\$:	MOVL	#1, STRING_LEN		: 0327
				50	D4	00031		CLRL	R0		: 0328
					05	00033		RSB			: 0330

; Routine Size: 52 bytes, Routine Base: _PAS\$CODE + 0080

: 270	0331	1	
: 271	0332	1	!<BLF/PAGE>

```
273 0333 1 %SBTTL 'PAS$GET_INTEGER - Find a signed number string'
274 0334 1 GLOBAL ROUTINE PAS$GET_INTEGER (      Get signed number string
275 0335 1      PFV: REF $PASSPFV FILE VARIABLE,      Pascal File Variable
276 0336 1      IN_FCB: REF $PASSFCB_CONTROL_BLOCK;      File control block
277 0337 1      STRING_ADDR,      Output string address
278 0338 1      STRING_LEN,      Output string length
279 0339 1      FCB: REF $PASSFCB_CONTROL_BLOCK      File control block
280 0340 1      ) : JSB_READ_UTIL =
281 0341 1
282 0342 1 ++
283 0343 1 FUNCTIONAL DESCRIPTION:
284 0344 1
285 0345 1      This procedure advances the textfile referenced by FCB until it
286 0346 1      locates a string that satisfies the Pascal INTEGER datatype
287 0347 1      syntax. The address and length of that string are returned as
288 0348 1      output parameters.
289 0349 1
290 0350 1 CALLING SEQUENCE:
291 0351 1
292 0352 1      Valid.wc.v = JSB PAS$GET_INTEGER (PFV.mr.r, IN_FCB.mr.r;
293 0353 1      STRING_ADDR.wl.v, STRING_LEN.wl.v, FCB.mr.r)
294 0354 1
295 0355 1 FORMAL PARAMETERS:
296 0356 1
297 0357 1      PFV      - Pascal File Variable of the file.
298 0358 1
299 0359 1      IN_FCB   - The File Control Block of the file being scanned.
300 0360 1      It is assumed to be a textfile.
301 0361 1
302 0362 1      STRING_ADDR - Output register parameter which is set to the
303 0363 1      address of the first byte of the string.
304 0364 1
305 0365 1      STRING_LEN - Output register parameter which is set to the
306 0366 1      length of the string in bytes.
307 0367 1
308 0368 1      FCB      - Output register parameter which is the same as IN_FCB.
309 0369 1
310 0370 1 IMPLICIT INPUTS:
311 0371 1
312 0372 1      It is assumed that lazy-lookahead is not in progress.
313 0373 1
314 0374 1 IMPLICIT OUTPUTS:
315 0375 1
316 0376 1      FCB$A_RECORD_CUR points to the next character after the string, or
317 0377 1      EOL.
318 0378 1
319 0379 1 ROUTINE VALUE:
320 0380 1
321 0381 1      NONE
322 0382 1
323 0383 1 SIDE EFFECTS:
324 0384 1
325 0385 1      1 if string is a valid integer, 0 otherwise.
326 0386 1      If failure is returned, STRING_LEN includes the first bad character.
327 0387 1
328 0388 1 SIGNALLED ERRORS:
329 0389 1
```



```
330 0390 1 ! NONE
331 0391 1 !
332 0392 1 !--
333 0393 1 !
334 0394 2 BEGIN
335 0395 2
336 0396 2 LOCAL
337 0397 2 CHAR, ! Character read
338 0398 2 VALID; ! 1 if string a valid unsigned.
339 0399 2
340 0400 2 !+
341 0401 2 ! Declare CHAR_BYTE which is the same as CHAR except that we can
342 0402 2 ! test it as a signed byte. We want to leave CHAR as a longword
343 0403 2 ! so that it can be used efficiently as an index.
344 0404 2 !-
345 0405 2
346 0406 2 BIND
347 0407 2 CHAR_BYTE = CHAR: BYTE SIGNED;
348 0408 2
349 0409 2 !+
350 0410 2 ! Find first character that is not a blank or a tab, possibly skipping
351 0411 2 ! records.
352 0412 2 !-
353 0413 2
354 0414 2 CHAR = FIND_NON_BLANK (PFV [PFV$R_PFV], IN_FCB [FCB$R_FCB]; FCB);
355 0415 2
356 0416 2 !+
357 0417 2 ! Initially, string is invalid.
358 0418 2 !-
359 0419 2
360 0420 2 VALID = 0;
361 0421 2
362 0422 2 !+
363 0423 2 ! At this point, CHAR contains the first character which is not a blank
364 0424 2 ! or a tab. Initialize STRING_ADDR.
365 0425 2 !-
366 0426 2
367 0427 2 STRING_ADDR = .FCB [FCB$A_RECORD_CUR];
368 0428 2
369 0429 2 !+
370 0430 2 ! If first character is a sign, advance pointer.
371 0431 2 !-
372 0432 2
373 0433 2 IF .CHAR_BYTE GEQ 0
374 0434 2 THEN
375 0435 2 IF .CLASSTAB [.CHAR] EQLU CLASS_SI
376 0436 2 THEN
377 0437 2 BEGIN
378 0438 2 FCB [FCB$A_RECORD_CUR] = .FCB [FCB$A_RECORD_CUR] + 1;
379 0439 2 IF .FCB [FCB$A_RECORD_CUR] LSSA .FCB [FCB$A_RECORD_END]
380 0440 2 THEN
381 0441 2 CHAR = CH$RCHAR (.FCB [FCB$A_RECORD_CUR])
382 0442 2 ELSE
383 0443 2 CHAR = %C' '; ! End of line
384 0444 2 END;
385 0445 2
386 0446 2 !+
```

```
387 0447 2 ! In a loop, classify the characters until end-of-line or an invalid
388 0448 ! character is found.
389 0449 !
390 0450 !
391 0451 WHILE 1 DO
392 0452 BEGIN
393 0453
394 0454 !+
395 0455 ! If the character's value is greater than or equal to 128,
396 0456 ! it can't possibly be valid, so exit. Do this by a test for
397 0457 ! negative on CHAR_BYTE.
398 0458 !-
399 0459
400 0460 IF .CHAR_BYTE LSS 0
401 0461 THEN
402 0462 EXITLOOP;
403 0463
404 0464 !+
405 0465 ! If the character is not a digit, exit.
406 0466 !-
407 0467
408 0468 IF .CLASSTAB [.CHAR] NEQU CLASS_DG
409 0469 THEN
410 0470 EXITLOOP;
411 0471
412 0472 !+
413 0473 ! At least one digit seen, so indicate string valid.
414 0474 !-
415 0475
416 0476 VALID = 1;
417 0477
418 0478 !+
419 0479 ! Get another character if not at end-of-line.
420 0480 !-
421 0481
422 0482 FCB [FCB$A_RECORD_CUR] = .FCB [FCB$A_RECORD_CUR] + 1;
423 0483 IF .FCB [FCB$A_RECORD_CUR] LSSA .FCB [FCB$A_RECORD_END]
424 0484 THEN
425 0485 CHAR = CH$RCHAR (.FCB [FCB$A_RECORD_CUR])
426 0486 ELSE
427 0487 EXITLOOP;
428 0488
429 0489 END; ! Of WHILE loop
430 0490
431 0491 !+
432 0492 ! Set STRING_LEN to length of string and return.
433 0493 !-
434 0494
435 0495 STRING_LEN = .FCB [FCB$A_RECORD_CUR] - .STRING_ADDR;
436 0496 IF .STRING_LEN EQL 0 ! If so, VALID must be zero
437 0497 THEN
438 0498 STRING_LEN = 1;
439 0499 RETURN (.VALID);
440 0500
441 0501 END; ! End of routine PAS$$GET_INTEGER
```



```
0000V 30 00000 PAS$$GET_INTEGER::
      51      EC      52 D4 00003      BSBW      FIND_NON_BLANK      : 0414
      54      61 A7 9E 00005      CLRL      VALID      : 0420
      50      61 D0 00009      MOVAB      -20(FCB), R1      : 0427
      13      50 95 0000C      MOVL      (R1), STRING_ADDR      :
      04      FF37 CF40 91 00010      TSTB      CHAR_BYTE      : 0433
      08      13 19 0000E      BLSS      1$      :
      61 D6 00018      CMPB      CLASSTAB[CHAR], #4      : 0435
      61 D1 0001A      BNEQ      1$      :
      1A 1F 0001E      INCL      (R1)      : 0438
      50      20 D0 00020      CMPL      (R1), -16(FCB)      : 0439
      50      50 95 00023 1$:      BLSSU      2$      :
      19      19 00025      MOVL      #32, CHAR      : 0443
      02      FF20 CF40 91 00027      TSTB      CHAR_BYTE      : 0460
      11      12 0002D      BLSS      3$      :
      52      01 D0 0002F      CMPB      CLASSTAB[CHAR], #2      : 0468
      61 D6 00032      BNEQ      3$      :
      61 D1 00034      MOVL      #1, VALID      : 0476
      06 1E 00038      INCL      (R1)      : 0482
      50      00 B1 9A 0003A 2$:      CMPL      (R1), -16(FCB)      : 0483
      55      61 54 C3 00040 3$:      BGEQU      3$      :
      03      12 00044      MOVZBL      @0(R1), CHAR      : 0485
      55      01 D0 00046      BRB      1$      :
      50      52 D0 00049 4$:      SUBL3      STRING_ADDR, (R1), STRING_LEN      : 0495
      05 0004C      BNEQ      4$      : 0496
      55      01 D0 00046      MOVL      #1, STRING_LEN      : 0498
      50      52 D0 00049 4$:      MOVL      VALID, R0      : 0499
      05 0004C      RSB      : 0501
```

; Routine Size: 77 bytes, Routine Base: _PAS\$CODE + 00B4

; 442 0502 1
; 443 0503 1 !<BLF/PAGE>

```
445 0504 1 XSBTTL 'PAS$$GET_REAL - Find a real number string'
446 0505 1 GLOBAL ROUTINE PAS$$GET_REAL (
447 0506 1     PFV: REF $PASS$PFV_FILE_VARIABLE,
448 0507 1     IN_FCB: REF $PASS$FCB_CONTROL_BLOCK;
449 0508 1     STRING_ADDR,
450 0509 1     STRING_LEN,
451 0510 1     FCB: REF $PASS$FCB_CONTROL_BLOCK
452 0511 1 ) : JSB_READ_UTIL =
453 0512 1
454 0513 1 ++
455 0514 1 FUNCTIONAL DESCRIPTION:
456 0515 1
457 0516 1     This procedure advances the textfile referenced by FCB until it
458 0517 1     locates a string that satisfies the Pascal REAL datatype
459 0518 1     syntax. The address and length of that string are returned as
460 0519 1     output parameters.
461 0520 1
462 0521 1 CALLING SEQUENCE:
463 0522 1
464 0523 1     Valid.wc.v = JSB PAS$$GET_REAL (PFV.mr.r, IN_FCB.mr.r;
465 0524 1     STRING_ADDR.wl.v, STRING_LEN.wl.v, FCB.mr.r)
466 0525 1
467 0526 1 FORMAL PARAMETERS:
468 0527 1
469 0528 1     PFV           - Pascal File Variable for the file.
470 0529 1
471 0530 1     IN_FCB        - The File Control Block of the file being scanned.
472 0531 1     It is assumed to be a textfile.
473 0532 1
474 0533 1     STRING_ADDR   - Output register parameter which is set to the
475 0534 1     address of the first byte of the string.
476 0535 1
477 0536 1     STRING_LEN    - Output register parameter which is set to the
478 0537 1     length of the string in bytes.
479 0538 1
480 0539 1     FCB           - Output register parameter which is the same as IN_FCB.
481 0540 1
482 0541 1 IMPLICIT INPUTS:
483 0542 1
484 0543 1     It is assumed that lazy-lookahead is not in progress.
485 0544 1
486 0545 1 IMPLICIT OUTPUTS:
487 0546 1
488 0547 1     FCB$A_RECORD_CUR points to the next character after the string, or
489 0548 1     EOL.
490 0549 1
491 0550 1 ROUTINE VALUE:
492 0551 1
493 0552 1     1 if string is a valid real, 0 otherwise.
494 0553 1     If failure is returned, STRING_LEN includes the first bad character
495 0554 1
496 0555 1 SIDE EFFECTS:
497 0556 1
498 0557 1     NONE
499 0558 1
500 0559 1 SIGNALLED ERRORS:
501 0560 1
```



```

502 0561 1 | NONE
503 0562 1 |
504 0563 1 |--
505 0564 1 |
506 0565 2 BEGIN
507 0566 2
508 0567 2 LOCAL
509 0568 2 CHAR, | Character read
510 0569 2 FLAGS: BITVECTOR [5]; | Indicate value fields seen
511 0570 2
512 0571 2 |
513 0572 2 | +
514 0573 2 | Declare CHAR_BYTE which is the same as CHAR except that we can
515 0574 2 | test it as a signed byte. We want to leave CHAR as a longword
516 0575 2 | so that it can be used efficiently as an index.
517 0576 2 |
518 0577 2 BIND
519 0578 2 CHAR_BYTE = CHAR: BYTE SIGNED;
520 0579 2
521 0580 2 LITERAL
522 0581 2 FLAGS_EXPLT = 0, | Exponent letter seen
523 0582 2 FLAGS_POINT = 1, | Decimal point seen
524 0583 2 FLAGS_FRADG = 2, | Fraction digit seen
525 0584 2 FLAGS_EXPDG = 3, | Exponent digit seen
526 0585 2 FLAGS_EXPSI = 4; | Exponent sign seen
527 0586 2
528 0587 2 |
529 0588 2 | +
530 0589 2 | Find first character that is not a blank or a tab, possibly skipping
531 0590 2 | records.
532 0591 2 |
533 0592 2 CHAR = FIND_NON_BLANK (PFV [PFV$R_PFV], IN_FCB [FCB$R_FCB]; FCB);
534 0593 2
535 0594 2 |
536 0595 2 | +
537 0596 2 | Initialize local flags.
538 0597 2 |
539 0598 2 FLAGS = 0;
540 0599 2
541 0600 2 |
542 0601 2 | +
543 0602 2 | At this point, CHAR contains the first character which is not a blank
544 0603 2 | or a tab. Initialize STRING_ADDR.
545 0604 2 |
546 0605 2 STRING_ADDR = .FCB [FCB$A_RECORD_CUR];
547 0606 2
548 0607 2 |
549 0608 2 | +
550 0609 2 | If first character is a sign, advance pointer.
551 0610 2 |
552 0611 2 IF .CHAR_BYTE GEQ 0
553 0612 2 THEN
554 0613 2 IF .CLASSTAB [.CHAR] EQLU CLASS_SI
555 0614 2 THEN
556 0615 2 BEGIN
557 0616 2 FCB [FCB$A_RECORD_CUR] = .FCB [FCB$A_RECORD_CUR] + 1;
558 0617 2 IF .FCB [FCB$A_RECORD_CUR] LSSA .FCB [FCB$A_RECORD_END]
```

```
559 0618 3      THEN
560 0619 3      CHAR = CH$RCHAR (.FCB [FCB$A_RECORD_CUR])
561 0620 3      ELSE
562 0621 3      CHAR = %C' ';    ! End of line
563 0622 3      END;
564 0623 3
565 0624 3
566 0625 3      !+ In a loop, classify the characters until end-of-line or an invalid
567 0626 3      ! character is found.
568 0627 3      !-
569 0628 3
570 0629 3      WHILE 1 DO
571 0630 3      BEGIN
572 0631 3
573 0632 3      !+
574 0633 3      ! If the character's value is greater than or equal to 128,
575 0634 3      ! it can't possibly be valid, so exit. Do this with a signed test
576 0635 3      ! on CHAR_BYTE.
577 0636 3      !-
578 0637 3
579 0638 3      IF .CHAR_BYTE LSS 0
580 0639 3      THEN
581 0640 3      EXITLOOP;
582 0641 3
583 0642 3      !+
584 0643 3      ! Select action based on character class.
585 0644 3      !-
586 0645 3
587 0646 3      CASE .CLASSTAB [.CHAR] FROM CLASS_IV TO CLASS_LT OF
588 0647 3
589 0648 3      SET
590 0649 3
591 0650 3      [CLASS_DG]:      ! Digit, always valid
592 0651 4      BEGIN
593 0652 4      IF .FLAGS [FLAGS_EXPLT]      ! Exponent letter already seen?
594 0653 4      THEN
595 0654 5      BEGIN
596 0655 5      FLAGS [FLAGS_EXPSI] = 1;      ! Prohibit future signs
597 0656 5      FLAGS [FLAGS_EXPDG] = 1;      ! Mark exponent digit seen
598 0657 5      END
599 0658 4      ELSE
600 0659 4      FLAGS [FLAGS_FRADG] = 1;      ! Mark fraction digit seen
601 0660 3      END;
602 0661 3
603 0662 3      [CLASS_SI]:      ! Sign character
604 0663 4      BEGIN
605 0664 4      IF NOT .FLAGS [FLAGS_EXPLT]      ! Exponent letter not seen?
606 0665 4      THEN
607 0666 4      EXITLOOP;      ! If so, invalid
608 0667 4      IF .FLAGS [FLAGS_EXPSI]      ! Exponent sign seen?
609 0668 4      THEN
610 0669 4      EXITLOOP;      ! If so, invalid
611 0670 4      FLAGS [FLAGS_EXPSI] = 1;      ! Indicate exponent sign seen
612 0671 3      END;
613 0672 3
614 0673 3      [CLASS_EL]:      ! Exponent letter
615 0674 4      BEGIN
```



```

: 616      0675 4      IF .FLAGS [FLAGS_EXPLT]      ! Exponent letter already seen?
: 617      0676 4      THEN
: 618      0677 4          EXITLOOP;                ! If so, invalid
: 619      0678 4      IF NOT .FLAGS [FLAGS_FRADG]    ! Fraction digit seen?
: 620      0679 4      THEN
: 621      0680 4          EXITLOOP;                ! If not, invalid
: 622      0681 4      FLAGS [FLAGS_EXPLT] = 1;        ! Mark exponent letter seen
: 623      0682 4      FLAGS [FLAGS_POINT] = 1;        ! Prohibit future decimal point
: 624      0683 4      END;
: 625      0684 3
: 626      0685 3      [CLASS_DP]:                ! Decimal point
: 627      0686 4      BEGIN
: 628      0687 4      IF .FLAGS [FLAGS_POINT]        ! Decimal point already seen?
: 629      0688 4      THEN
: 630      0689 4          EXITLOOP;                ! If so, invalid
: 631      0690 4      FLAGS [FLAGS_POINT] = 1;        ! Mark decimal point seen
: 632      0691 4      END;
: 633      0692 3
: 634      0693 3      [INRANGE, OTRANGE]:
: 635      0694 3      EXITLOOP;                ! Invalid
: 636      0695 3
: 637      0696 3      TES;
: 638      0697 3
: 639      0698 3
: 640      0699 3      !+ Get another character if not at end-of-line.
: 641      0700 3      !-
: 642      0701 3
: 643      0702 3      FCB [FCB$A_RECORD_CUR] = .FCB [FCB$A_RECORD_CUR] + 1;
: 644      0703 3      IF .FCB [FCB$A_RECORD_CUR] LSSA .FCB [FCB$A_RECORD_END]
: 645      0704 3      THEN
: 646      0705 3          CHAR = CH$RCHAR (.FCB [FCB$A_RECORD_CUR])
: 647      0706 3      ELSE
: 648      0707 3          EXITLOOP;
: 649      0708 3
: 650      0709 2      END;                ! Of WHILE loop
: 651      0710 2
: 652      0711 2      !+
: 653      0712 2      ! Set STRING_LEN to length of string and return function value indicating
: 654      0713 2      ! whether or not string is valid.
: 655      0714 2      !-
: 656      0715 2
: 657      0716 2      STRING_LEN = .FCB [FCB$A_RECORD_CUR] - .STRING_ADDR;
: 658      0717 2      IF .STRING_LEN EQL 0                ! If so, string is invalid
: 659      0718 2      THEN
: 660      0719 2          STRING_LEN = 1;
: 661      0720 3      RETURN (
: 662      0721 3          IF .FLAGS [FLAGS_FRADG] AND      ! Fraction digit required
: 663      0722 4          ((NOT .FLAGS [FLAGS_EXPLT]) OR .FLAGS [FLAGS_EXPDG]) ! If exponent, must have digits
: 664      0723 3      THEN
: 665      0724 3          1      ! Valid
: 666      0725 3      ELSE
: 667      0726 3          0      ! Invalid
: 668      0727 3      );
: 669      0728 2
: 670      0729 1      END;                ! End of routine PASS$GET_REAL
```

0000V 30 00000 PASS\$GET REAL::						
		52	94 00003	BSBW	FIND_NON_BLANK	: 0592
		51	EC A7 9E 00005	CLRB	FLAGS	: 0598
		54	61 D0 00009	MOVAB	-20(FCB), R1	: 0605
			50 95 0000C	MOVL	(R1), STRING_ADDR	
			13 19 0000E	TSTB	CHAR_BYTE	: 0611
		04	FEEA CF40 91 00010	BLSS	1\$: 0613
			0B 12 00016	CMPB	CLASSTAB[CHAR], #4	
			61 D6 00018	BNEQ	1\$: 0616
		F0 A7	61 D1 0001A	INCL	(R1)	: 0617
			52 1F 0001E	CMPL	(R1), -16(FCB)	
		50	20 D0 00020	BLSSU	10\$: 0621
			50 95 00023 1\$:	MOVL	#32, CHAR	: 0638
			51 19 00025	TSTB	CHAR_BYTE	
			FED3 CF40 8F 00027	BLSS	11\$: 0646
0035	06	00	004A 0002E 2\$:	CASEB	CLASSTAB[CHAR], #0, #6	
	0010	004A	001D 00036	.WORD	11\$-2\$,-	
					11\$-2\$,-	
					3\$-2\$,-	
					7\$-2\$,-	
					5\$-2\$,-	
					6\$-2\$,-	
					11\$-2\$	
			3A 11 0003C	BRB	11\$: 0694
		05	52 E9 0003E 3\$:	BLBC	FLAGS, 4\$: 0652
		52	18 88 00041	BISB2	#24, FLAGS	: 0656
			24 11 00044	BRB	9\$: 0652
		52	04 88 00046 4\$:	BISB2	#4, FLAGS	: 0659
			1F 11 00049	BRB	9\$: 0646
		2A	52 E9 0004B 5\$:	BLBC	FLAGS, 11\$: 0664
26		52	04 E0 0004E	BBS	#4, FLAGS, 11\$: 0667
		52	10 88 00052	BISB2	#16, FLAGS	: 0670
			13 11 00055	BRB	9\$: 0646
		1E	52 E8 00057 6\$:	BLBS	FLAGS, 11\$: 0675
1A		52	02 E1 0005A	BBC	#2, FLAGS, 11\$: 0678
		52	01 88 0005E	BISB2	#1, FLAGS	: 0681
			04 11 00061	BRB	8\$: 0682
11		52	01 E0 00063 7\$:	BBS	#1, FLAGS, 11\$: 0687
		52	02 88 00067 8\$:	BISB2	#2, FLAGS	: 0690
			61 D6 0006A 9\$:	INCL	(R1)	: 0702
		F0 A7	61 D1 0006C	CMPL	(R1), -16(FCB)	: 0703
			06 1E 00070	BGEQU	11\$	
		50	B1 9A 00072 10\$:	MOVZBL	@0(R1), CHAR	: 0705
			AB 11 00076	BRB	1\$	
55		61	54 C3 00078 11\$:	SUBL3	STRING_ADDR, (R1), STRING_LEN	: 0716
			03 12 0007C	BNEQ	12\$: 0717
		55	01 D0 0007E	MOVL	#1, STRING_LEN	: 0719
0B		52	02 E1 00081 12\$:	BBC	#2, FLAGS, -14\$: 0721
		04	52 E9 00085	BLBC	FLAGS, 13\$: 0722
04		52	03 E1 00088	BBC	#3, FLAGS, 14\$	
			01 D0 0008C 13\$:	MOVL	#1, R0	: 0721
			05 0008F	RSB		
		50	D4 00090 14\$:	CLRL	R0	: 0729
			05 00092	RSB		

PASS\$READ_UTIL Utility routines used by READ
1-001 PASS\$GET_REAL - Find a real number string

C 6
16-Sep-1984 01:55:25
14-Sep-1984 12:51:47

VAX-11 Bliss-32 V4.0-742
[PASRTL.SRC]PASREADUT.B32;1

Page 17
(5)

; Routine Size: 147 bytes, Routine Base: _PASS\$CODE + 0101

: 671 0730 1
: 672 0731 1 !<BLF/PAGE>

```

674 0732 1 %SBTTL 'PAS$$GET_ENUMERATED - Find an enumerated value string'
675 0733 1 GLOBAL ROUTINE PAS$$GET_ENUMERATED (      Get enumerated value string
676 0734 1      PFV: REF $PASS$PFV FILE_VARIABLE,      Pascal File Variable
677 0735 1      IN_FCB: REF $PASS$FCB_CONTROL_BLOCK;      File control block
678 0736 1      STRING_ADDR,      Output string address
679 0737 1      STRING_LEN,      Output string length
680 0738 1      FCB: REF $PASS$FCB_CONTROL_BLOCK      File control block
681 0739 1      ) : JSB_READ_UTIL =
682 0740 1
683 0741 1 ++
684 0742 1 FUNCTIONAL DESCRIPTION:
685 0743 1
686 0744 1     This procedure advances the textfile referenced by FCB until it
687 0745 1     locates a string that satisfies the Pascal enumerated type value
688 0746 1     syntax. The address and length of that string are returned as
689 0747 1     output parameters.
690 0748 1
691 0749 1 CALLING SEQUENCE:
692 0750 1
693 0751 1     Valid.wc.v = JSB PAS$$GET_ENUMERATED (PFV.mr.r, IN_FCB.mr.r;
694 0752 1     STRING_ADDR.wl.v, STRING_LEN.wl.v, FCB.mr.r)
695 0753 1
696 0754 1 FORMAL PARAMETERS:
697 0755 1
698 0756 1     PFV          - Pascal File Variable for the file.
699 0757 1
700 0758 1     IN_FCB       - The File Control Block of the file being scanned.
701 0759 1     It is assumed to be a textfile.
702 0760 1
703 0761 1     STRING_ADDR  - Output register parameter which is set to the
704 0762 1     address of the first byte of the string.
705 0763 1
706 0764 1     STRING_LEN   - Output register parameter which is set to the
707 0765 1     length of the string in bytes.
708 0766 1
709 0767 1     FCB          - Output register parameter which is the same as IN_FCB.
710 0768 1
711 0769 1 IMPLICIT INPUTS:
712 0770 1
713 0771 1     It is assumed that lazy-lookahead is not in progress.
714 0772 1
715 0773 1 IMPLICIT OUTPUTS:
716 0774 1
717 0775 1     FCB$A_RECORD_CUR points to the next character after the string, or
718 0776 1     EOL.
719 0777 1
720 0778 1 ROUTINE VALUE:
721 0779 1
722 0780 1     1 if string is a valid enumerated value, 0 otherwise
723 0781 1     If failure is returned, STRING_LEN includes the first bad character
724 0782 1
725 0783 1 SIDE EFFECTS:
726 0784 1
727 0785 1     NONE
728 0786 1
729 0787 1 SIGNALLED ERRORS:
730 0788 1
```



```

731 0789 1 1 NONE
732 0790 1 1
733 0791 1 1 --
734 0792 1 1
735 0793 2 BEGIN
736 0794 2
737 0795 2 LOCAL
738 0796 2 CHAR,
739 0797 2 VALID_CHAR_MASK: BITVECTOR [32];
740 0798 2
741 0799 2
742 0C00 2
743 0801 2
744 0802 2
745 0803 2
746 0804 2
747 0805 2
748 0806 2
749 0807 2
750 0808 2
751 0809 2
752 0810 2
753 0811 2
754 0812 2
755 0813 2
756 0814 2
757 0815 2
758 0816 2
759 0817 2
760 0818 2
761 0819 2
762 0820 2
763 0821 2
764 0822 2
765 0823 2
766 0824 2
767 0825 2
768 0826 2
769 0827 2
770 0828 2
771 0829 2
772 0830 2
773 0831 2
774 0832 2
775 0833 2
776 0834 2
777 0835 2
778 0836 2
779 0837 2
780 0838 2
781 0839 2
782 0840 2
783 0841 2
784 0842 2
785 0843 2
786 0844 2
787 0845 2

      NONE

      --

      BEGIN

      LOCAL
      CHAR,
      VALID_CHAR_MASK: BITVECTOR [32];

      ! Character read
      ! Bit is set if associated
      ! character class is valid
      ! at this point.

      +
      Declare CHAR_BYTE which is the same as CHAR except that we can
      test it as a signed byte. We want to leave CHAR as a longword
      so that it can be used efficiently as an index.
      -

      BIND
      CHAR_BYTE = CHAR: BYTE SIGNED;

      +
      Find first character that is not a blank or a tab, possibly skipping
      records.
      -

      CHAR = FIND_NON_BLANK (PFV [PFV$R_PFV], IN_FCB [FCB$R_FCB]; FCB);

      +
      At this point, CHAR contains the first character which is not a blank
      or a tab. Initialize STRING_ADDR.
      -

      STRING_ADDR = .FCB [FCB$A_RECORD_CUR];

      +
      First character must be a letter. (Class LT excludes exponent
      letters, so add class EL).
      -

      VALID_CHAR_MASK = (1^CLASS_LT)+(1^CLASS_EL);

      +
      In a loop, classify the characters until end-of-line or an invalid
      character is found.
      -

      WHILE 1 DO
      BEGIN
      +
      If the character's value is greater than or equal to 128,
      it can't possibly be valid, so exit. Do this with a signed test
      on CHAR_BYTE.
      -

```

```

: 788      0846      3      IF .CHAR_BYTE LSS 0
: 789      0847      3      THEN
: 790      0848      3          EXITLOOP;
: 791      0849      3
: 792      0850      3      !+
: 793      0851      3      ! Get the class of the character from CLASSTAB and test its
: 794      0852      3      ! corresponding bit in VALID_CHAR_MASK. If it is not set, that
: 795      0853      3      ! character is not acceptable.
: 796      0854      3      !-
: 797      0855      3
: 798      0856      3      IF NOT .VALID_CHAR_MASK [.CLASSTAB [.CHAR]]
: 799      0857      3      THEN
: 800      0858      3          EXITLOOP;
: 801      0859      3
: 802      0860      3      !+
: 803      0861      3      ! Allow digits to appear from now on.
: 804      0862      3      !-
: 805      0863      3
: 806      0864      3      VALID_CHAR_MASK [CLASS_DG] = 1;
: 807      0865      3
: 808      0866      3      !+
: 809      0867      3      ! Get another character if not at end-of-line.
: 810      0868      3      !-
: 811      0869      3
: 812      0870      3      FCB [FCB$A_RECORD_CUR] = .FCB [FCB$A_RECORD_CUR] + 1;
: 813      0871      3      IF .FCB [FCB$A_RECORD_CUR] LSSA .FCB [FCB$A_RECORD_END]
: 814      0872      3      THEN
: 815      0873      3          CHAR = CH$RCHAR (.FCB [FCB$A_RECORD_CUR])
: 816      0874      3      ELSE
: 817      0875      3          EXITLOOP;
: 818      0876      3
: 819      0877      3      END;      ! Of WHILE loop
: 820      0878      3
: 821      0879      3
: 822      0880      3      !+
: 823      0881      3      ! Set STRING_LEN to length of string and return function value indicating
: 824      0882      3      ! whether or not string is valid.
: 825      0883      3      !-
: 826      0884      3
: 827      0885      3      STRING_LEN = .FCB [FCB$A_RECORD_CUR] - .STRING_ADDR;
: 828      0886      3      IF .STRING_LEN NEQ 0
: 829      0887      3      THEN
: 830      0888      3          RETURN 1;
: 831      0889      3
: 832      0890      3      STRING_LEN = 1;      ! Include first had character
: 833      0891      3      RETURN 0;      ! Failure
: 834      0892      3
: 835      0893      3      END;
                                     ! End of routine PAS$$GET_ENUMERATED
```

0000V 30 0000 PAS\$\$GET ENUMERATED::

54 EC A7 D0 00003
51 60 8F 9A 00007BSBW FIND NON BLANK
MOVL -20(FCB), STRING_ADDR
MOVZBL #96, VALID_CHAR_MASK: 0815
: 0822
: 0830

PASS\$READ_UTIL
1-001

Utility routines used by READ
PASS\$GET_ENUMERATED - Find an enumerated value

G 6
16-Sep-1984 01:55:25
14-Sep-1984 12:51:47

VAX-11 Bliss-32 V4.0-742
[PASRTL.SRC]PASREADUT.B32;1

Page 21
(6)

			50	95	0000B	1\$:	TSTB	CHAR_BYTE	:	0846
			1D	19	0000D		BLSS	2\$:	
13	52	FE5B	CF40	9A	0000F		MOVZBL	CLASSTAB[CHAR], R2	:	0856
	51		52	E1	00015		BBC	R2, VALID_CHAR_MASK, 2\$:	
	51		04	88	00019		BISB2	#4, VALID_CHAR_MASK	:	0864
		EC	A7	D6	0001C		INCL	-20(FCB)	:	0870
	F0	A7	EC	A7	D1	0001F	CMP	-20(FCB), -16(FCB)	:	0871
			06	1E	00024		BGEQU	2\$:	
	50		EC	B7	9A	00026	MOVZBL	@-20(FCB), CHAR	:	0873
			DF	11	0002A		BRB	1\$:	
55	EC	A7	54	C3	0002C	2\$:	SUBL3	STRING_ADDR, -20(FCB), STRING_LEN	:	0885
			04	13	00031		BEQL	3\$:	0886
	50		01	D0	00033		MOVL	#1, R0	:	0888
				05	00036		RSB		:	
	55		01	D0	00037	3\$:	MOVL	#1, STRING_LEN	:	0890
			50	D4	0003A		CLRL	R0	:	0891
				05	0003C		RSB		:	0893

; Routine Size: 61 bytes, Routine Base: _PASS\$CODE + 0194

: 836 0894 1
: 837 0895 1 !<BLF/PAGE>

```

: 839 0896 1 %SBTTL 'FIND_NON_BLANK - Find first non-blank'
: 840 0897 1 ROUTINE FIND_NON_BLANK (
: 841 0898 1 PFV: REF $PASSPFV FILE VARIABLE,
: 842 0899 1 IN_FCB: REF $PASSFCB CONTROL_BLOCK;
: 843 0900 1 FCB: REF $PASSFCB CONTROL_BLOCK
: 844 0901 1 ) : JSB_FIND_NON_BLANK =
: 845 0902 1
: 846 0903 1 ++
: 847 0904 1 FUNCTIONAL DESCRIPTION:
: 848 0905 1
: 849 0906 1 This procedure advances the textfile referenced by FCB until it
: 850 0907 1 locates the first character which is not a blank or a tab. It
: 851 0908 1 returns that character as its function value.
: 852 0909 1
: 853 0910 1 CALLING SEQUENCE:
: 854 0911 1
: 855 0912 1 CHAR.wt.v = JSB_FIND_NON_BLANK (PFV.mr.r, IN_FCB.mr.r; FCB.mr.r)
: 856 0913 1
: 857 0914 1 FORMAL PARAMETERS:
: 858 0915 1
: 859 0916 1 PFV - Pascal File Variable for the file.
: 860 0917 1
: 861 0918 1 IN_FCB - The File Control Block of the file being scanned.
: 862 0919 1 It is assumed to be a textfile.
: 863 0920 1
: 864 0921 1 FCB - Output register parameter which is the same as IN_FCB.
: 865 0922 1
: 866 0923 1 IMPLICIT INPUTS:
: 867 0924 1
: 868 0925 1 It is assumed that lazy-lookahead is not in progress.
: 869 0926 1
: 870 0927 1 IMPLICIT OUTPUTS:
: 871 0928 1
: 872 0929 1 FCB$A_RECORD_CUR points to the found character.
: 873 0930 1
: 874 0931 1 ROUTINE VALUE:
: 875 0932 1
: 876 0933 1 The character found which is not a blank or a tab.
: 877 0934 1
: 878 0935 1 SIDE EFFECTS:
: 879 0936 1
: 880 0937 1 NONE
: 881 0938 1
: 882 0939 1 SIGNALLED ERRORS:
: 883 0940 1
: 884 0941 1 GETAFTEOF - GET after end-of-file
: 885 0942 1
: 886 0943 1 --
: 887 0944 1
: 888 0945 2 BEGIN
: 889 0946 2
: 890 0947 2 LOCAL
: 891 0948 2 CHAR; ! Character read
: 892 0949 2
: 893 0950 2 !+
: 894 0951 2 ! Declare CHAR_BYTE which is the same as CHAR except that we can
: 895 0952 2 ! test it as a signed byte. We want to leave CHAR as a longword
```



```
896 0953 2 | so that it can be used efficiently as an index.
897 0954 2 |
898 0955 2
899 0956 2 BIND
900 0957 2 CHAR_BYTE = CHAR: BYTE SIGNED;
901 0958 2
902 0959 2 FCB = .IN_FCB;
903 0960 2
904 0961 2 |
905 0962 2 | Find first character that is not a blank or a tab, possibly skipping
906 0963 2 | records.
907 0964 2 |
908 0965 2
909 0966 2 WHILE 1 DO
910 0967 2 BEGIN
911 0968 2 |
912 0969 2 | If we are at end-of-line, get another record. This is done by
913 0970 2 | setting lazy-lookahead and then calling PAS$$LOOK_AHEAD.
914 0971 2 |
915 0972 2 |
916 0973 2 IF .FCB [FCB$A_RECORD_CUR] GEQA .FCB [FCB$A_RECORD_END]
917 0974 2 THEN
918 0975 2 BEGIN
919 0976 2 FCB [FCB$V_LAZY] = 1; ! Set lazy lookahead
920 0977 2 PAS$$LOOK_AHEAD (PFV [PFV$R_PFV], FCB [FCB$R_FCB]; FCB);
921 0978 2 END
922 0979 2 ELSE
923 0980 2 BEGIN
924 0981 2 |
925 0982 2 | Get next character, advancing pointer, and check class for blank
926 0983 2 | or tab.
927 0984 2 |
928 0985 2 CHAR = CH$RCHAR A (FCB [FCB$A_RECORD_CUR]);
929 0986 2 IF (.CHAR_BYTE [SS 0) OR (.CLASSTAB [.CHAR] NEQ CLASS_BT)
930 0987 2 THEN
931 0988 2 BEGIN
932 0989 2 |
933 0990 2 | Non blank/tab found. Reset record pointer to point
934 0991 2 | to character and exit loop.
935 0992 2 |
936 0993 2 FCB [FCB$A_RECORD_CUR] = .FCB [FCB$A_RECORD_CUR] - 1;
937 0994 2 EXITLOOP;
938 0995 2 END;
939 0996 2 END;
940 0997 2
941 0998 2 END; ! Of WHILE loop
942 0999 2
943 1000 2 RETURN .CHAR; ! Return found character
944 1001 2
945 1002 1 END; ! End of routine FIND_NON_BLANK
```

.EXTRN PAS\$\$LOOK_AHEAD

F0 A7 EC A7 D1 0000 FIND_NON_BLANK:
[MPL -20(FCB), -16(FCB)

; 0973

PASS\$READ_UTIL Utility routines used by READ
1-001 FIND_NON_BLANK - Find first non-blank

J 6
16-Sep-1984 01:55:25
14-Sep-1984 12:51:47

VAX-11 Bliss-32 V4.0-742
[PASRTL.SRC]PASREADUT.B32;1

Page 24
(7)

```

      FD  A7      00000000G  0C  1F 00005
                        04  88 00007
                        00  16 0000B
                        ED  11 00011
      58      EC  B7  9A 00013 1$:
                        EC  A7  D6 00017
                        58  95 0001A
                        08  19 0001C
      01      FE0C CF48 91 0001E
                        DA  13 00024
                        EC  A7  D7 00026 2$:
      50      58  D0 00029
                        05 0002C
```

```

BLSSU  1$
BISB2  #4, -3(FCB)
JSB    PASS$LOOK_AHEAD
BRB    FIND_NON_BLANK
MOVZBL @-20(FCB), CHAR
INCL   -20(FCB)
TSTB   CHAR_BYTE
BLSS   2$
CMPB   CLASSTAB[CHAR], #1
BEQL   FIND_NON_BLANK
DECL   -20(FCB)
MOVL   CHAR, R0
RSB
```

```

: 0976
: 0977
: 0973
: 0985
: 0986
:
:
: 0993
: 1000
: 1002
```

; Routine Size: 45 bytes, Routine Base: _PASS\$CODE + 01D1

```

: 946      1003 1
: 947      1004 1 !<BLF/PAGE>
```


PASS\$READ_UTIL Utility routines used by READ
1-001 FIND_NON_BLANK - Find first non-blank

K 6
16-Sep-1984 01:55:25 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:51:47 [PASRTL.SRC]PASREADUT.B32;1

Page 25
(8)

: 949 1005 1 END
: 950 1006 1
: 951 1007 0 ELUDOM

! End of module PASS\$READ_UTIL

PSECT SUMMARY

Name	Bytes	Attributes
_PASS\$CODE	510	NOVEC,NOWRT, RD , EXE, SHR, LCL, REL, CON, PIC,ALIGN(2)

Library Statistics

File	----- Total	Symbols Loaded	----- Percent	Pages Mapped	Processing Time
\$255\$DUA28:[SYSLIB]STARLET.L32;1	9776	0	0	581	00:01.0
\$255\$DUA28:[PASRTL.OBJ]PASLIB.L32;1	427	86	20	33	00:00.4

COMMAND QUALIFIERS

BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/NOTRACE/LIS=LIS\$:PASREADUT/OBJ=OBJ\$:PASREADUT MSRC\$:PASREADUT/UPDATE=(ENH\$:PASREADUT)

: Size: 382 code + 128 data bytes
: Run Time: 00:15.4
: Elapsed Time: 00:53.6
: Lines/CPU Min: 3923
: Lexemes/CPU-Min: 15841
: Memory Used: 110 pages
: Compilation Complete

0296

AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY